Combinatorics, Probability and Computing (2007) 16, 595–619. © 2006 Cambridge University Press
doi:10.1017/S0963548306008133 First published online 3 November 2006 Printed in the United Kingdom

Searching for a Black Hole in Synchronous Tree Networks[¶]

JUREK CZYZOWICZ,[1*] DARIUSZ KOWALSKI,[2†]
EURIPIDES MARKOU[3‡] and ANDRZEJ PELC[1§]

[1]Département d'Informatique, Université du Québec en Outaouais,
Gatineau, Québec J8X 3X7, Canada
(e-mail: {jurek, pelc}@uqo.ca)
[2]Department of Computer Science, The University of Liverpool,
Chadwick Building, Peach Street, Liverpool L69 7ZF, UK
(e–mail: D.R.Kowalski@csc.liv.ac.uk)
[3]Department of Informatics and Telecommunications, National Kapodistrian University of Athens,
Panepistimiopolis 15784, Athens, Greece
(e-mail: emarkou@di.uoa.gr)

Received 25 January 2005; revised 6 March 2006

A black hole is a highly harmful stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. We consider the task of locating a black hole in a (partially) synchronous tree network, assuming an upper bound on the time of any edge traversal by an agent. The minimum number of agents capable of identifying a black hole is two. For a given tree and given starting node we are interested in the fastest-possible black hole search by two agents. For arbitrary trees we give a 5/3-approximation algorithm for this problem. We give optimal black hole search algorithms for two 'extreme' classes of trees: the class of lines and the class of trees in which any internal node (including the root which is the starting node) has at least two children.

* Research supported in part by an NSERC grant.
† Research supported in part by grants from KBN (4T11C04425) and UE DELIS.
‡ This work was done during this author's stay at the Research Chair in Distributed Computing of the Université du Québec en Outaouais, as a postdoctoral fellow.
§ Research supported in part by an NSERC grant and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.
¶ A preliminary version of this paper (entitled 'Searching for a black hole in tree networks') appeared in the *Proc. 8th International Conference on Principles of Distributed Systems* (OPODIS'2004), December 2004, Grenoble, France.

1. Introduction

1.1. The background and the problem

The security of mobile agents working in a network environment is an important issue which has recently received increasing attention. Protecting agents from 'host attacks', *i.e.*, harmful items stored in nodes of the network, has become almost as urgent as protecting a host, *i.e.*, a node of the network, from an agent's attack [9, 10]. Various methods of protecting mobile agents against malicious hosts have been discussed, *e.g.*, in [5, 6, 8, 9, 10, 11].

In this paper we consider hostile hosts of a particularly harmful nature, called *black holes* [1, 2, 3, 4]. A black hole is a stationary process residing in a node of a network and destroying all mobile agents visiting the node, without leaving any trace. Since agents cannot avoid being annihilated once they visit a black hole, the only way of protection against such processes is identifying the hostile node and avoiding further visiting it. Hence we are dealing with the issue of locating a black hole: assuming that there is at most one black hole in the network, at least one surviving agent must find the location of the black hole if it exists, or answer that there is no black hole otherwise. The only way to locate the black hole is to visit it by at least one agent, hence, as observed in [2], at least two agents are necessary for one of them to locate the black hole and survive. Throughout the paper we assume that the number of agents is minimum possible for our task, *i.e.*, 2, and that they start from the same node, known to be safe.

In [1, 2, 3, 4] the issue of efficient black hole search was extensively studied in many types of networks. The underlying assumption in these papers was that the network is totally asynchronous, *i.e.*, while every edge traversal by a mobile agent takes finite time, there is no upper bound on this time. In this setting it was observed that, in order to solve the problem, the network must be 2-connected, in particular, black hole search is infeasible in trees. This is because in asynchronous networks it is impossible to distinguish a black hole from a 'slow' link incident to it. Hence the only way to locate a black hole is to visit all other nodes and learn that they are safe. (In particular, it is impossible to answer the question of whether a black hole actually exists in the network, hence [1, 2, 3, 4] worked under the assumption that there is exactly one black hole and the task was to locate it.)

Totally asynchronous networks rarely occur in practice. Often a (possibly large) upper bound on the time of traversing any edge by an agent can be established. Hence it is interesting to study black hole search in such partially synchronous networks. Without loss of generality, this upper bound on edge traversal time can be normalized to 1, which yields the following definition of the time of a black hole search scheme: this is the maximum time taken by the scheme, *i.e.*, the time under the worst-case location of the black hole (or when it does not exist in the network), assuming that all edge traversals take time 1.

Our partially synchronous scenario makes a dramatic change to the problem of searching for a black hole. Now it is possible to use the time-out mechanism to locate the black hole in any graph, with only two agents, as follows. Agents proceed along edges of a spanning tree. If they are at a safe node v, one agent goes to the adjacent node and returns, while the other agent waits at v. If after time 2 the first agent has not returned, the other one survives and knows the location of the black hole. Otherwise, the adjacent node is known to be safe and both agents can move to it. This is in fact a variant of

the *cautious walk* described in [2] but combining it with the time-out mechanism makes black hole search feasible in any graph. Hence the issue is now not the feasibility but the time efficiency of black hole search, and the present paper is devoted to this problem.

Since, for any network, black hole search can be done using only the edges of its spanning tree, solving the problem of fast black hole search on trees seems a natural first step. Hence, in this paper we restrict attention to black hole search in tree networks using two agents, and our goal is to accomplish this task in minimum time. Clearly, in many graphs, there are more efficient black hole search schemes than those operating in a spanning tree of the graph, and the generalization of our problem to arbitrary networks remains an important and interesting open issue.

The time of a black hole search scheme should be distinguished from the time complexity of the algorithm producing such a scheme. While the first was defined above for a given input consisting of a network and a starting node, and is in fact the larger of the numbers of time units spent by the two agents, the second is the time of producing such a scheme by the algorithm. In other words, the time of the scheme is the time of walking and the time complexity of the algorithm is the time of thinking.

Constructing a fastest black hole search scheme for arbitrary trees turns out to be far from trivial. In particular, the following problem remains open. Does there exist a polynomial time algorithm which, given a tree and a starting node as input, produces a black hole search scheme working in shortest-possible time for this input? Nevertheless, we show fastest schemes for some classes of trees and give a 5/3-approximation algorithm for the general case.

1.2. Our results

For arbitrary trees we give a 5/3-approximation algorithm for the black hole search problem. More precisely, given a tree and a starting node as input, our algorithm produces a black hole search scheme whose time is at most 5/3 of the shortest-possible time for this input.

We give optimal black hole search algorithms for two 'extreme' classes of trees: the class of lines and the class of trees in which any internal node (including the root which is the starting node) has at least 2 children. More precisely, for every input in the respective classes these algorithms produce a black hole search scheme whose time is the shortest possible for this input.

All our algorithms work in time linear in the size of the input.

2. Model and terminology

We consider a tree T rooted at node s which is the starting node of both agents, and is assumed to be safe (s is not a black hole). Notions of child, parent, descendant and ancestor are meant with respect to this rooted tree. The down degree of a node is the number of its children. Agents have distinct labels. They can communicate only when they meet (and not, *e.g.*, by leaving messages at nodes). We assume that there is at most one black hole in the network. This is a node which destroys any agents visiting it. A black hole search scheme (*BHS-scheme*) for the input (T, s) is a pair of sequences of edge traversals (moves) of each of the two agents, with the following properties.

- Each move takes one time unit.
- A move can be empty (this corresponds to an agent waiting at a node).
- Upon completion of the scheme there is at least one surviving agent, *i.e.*, an agent that has not visited the black hole, and this agent either knows the location of the black hole or knows that there is no black hole in the tree. The surviving agents must return to s.

The time of a black hole search scheme is the number of time units until the completion of the scheme, assuming the worst-case location of the black hole (or its absence, whichever is worse). It is easy to see that the worst case for a given scheme occurs when there is no black hole in the tree or when the black hole is the last unvisited node, both cases yielding the same time. A scheme is called *fastest* for a given input if its time is the shortest possible for this input.

For any edge of a tree we define the following states:

- *unknown*, if no agent has moved yet along this edge (initial state of every edge),
- *explored*, if either the remaining agents know that there is no black hole incident to this edge, or they know which end of the edge is a black hole.

Any BHS-scheme must have the following property. After a finite number of steps, at least one agent stays alive and all edges are explored (there is at most one black hole, so once the black hole has been found, all edges are explored).

The *explored territory* at step t of a BHS-scheme is the set of explored edges. At the beginning of a BHS-scheme the explored territory is empty. We say that a *meeting* occurs in node v at step t when the agents meet at node v and exchange information which *strictly increases* the explored territory. Node v is called a *meeting point* at step t.

Note that in between meetings, an edge may be neither unknown nor explored. This is the case when an unknown edge has been just traversed by an agent.

In any step of a BHS-scheme, an agent can traverse an edge or wait in a node. Also the two agents can meet. If at step t a meeting occurs, then the explored territory at step t is defined as the explored territory *after* the meeting. The sequence of steps of a BHS-scheme between two consecutive meetings is called a *phase*.

3. Preliminary results

Lemma 3.1. *In a BHS-scheme, an unexplored edge cannot be traversed by both agents.*

Proof. Suppose that an unexplored edge e has been traversed by an agent and while e remains unexplored (which means that the two agents have not yet met after the traversal of e), the other agent traverses it. If this edge is incident to a black hole, then both agents vanish, which means that this is not a BHS-scheme. □

Hence in a BHS-scheme, an edge can be explored only in the following way. An agent traverses this edge and then a meeting is scheduled. Whether it occurs or not (in the latter case the agent vanished in the black hole) the edge becomes explored.

Lemma 3.2. *During a phase of a BHS-scheme an agent can traverse at most* one *unexplored edge.*

Proof. Suppose that an agent traverses two unexplored edges. If one of these edges is incident to a black hole and hence the agent vanishes, then there is no way for the other agent to locate the black hole without vanishing, which means that this is not a BHS-scheme. □

Therefore an unknown edge could be explored in the next phase only if it is adjacent to the explored territory. The explored territory increases only at scheduled meeting points.

Lemma 3.3. *At the end of each phase, the explored territory is increased by one or two edges, or the black hole is found.*

Proof. By the end of a phase the explored territory is increased by at least one edge. By Lemma 3.2, an agent can traverse at most one unexplored edge during a phase; thus both agents can traverse a total of at most two unexplored edges during a phase. □

We define a *1-phase* to be a phase in which exactly one edge is explored. Similarly, we define a *2-phase* to be a phase in which exactly two edges are explored. In view of Lemma 3.3, every phase is either a 1-phase or a 2-phase.

Lemma 3.4. *Let v be a meeting point at step t in a BHS-scheme. Then at least one of the following holds: $v = s$ or v is an endpoint of an edge which was already explored at step $t-1$.*

Proof. Suppose that $v \neq s$ and every edge incident to v was unexplored at step $t-1$. Since v is a meeting point at step t, both agents are scheduled to be at v by that time. This means that at least one unexplored edge has been traversed by both agents, which by Lemma 3.1 is impossible. □

Hence an agent which traversed an unexplored edge must return to the explored territory in order to go to the meeting point. A corollary of Lemmas 3.1, 3.2 and 3.4 is that at any step of a BHS-scheme the explored territory is connected.

A node p is called a *limit* of the explored territory at step t if it is incident both to an explored and to an unexplored edge, or if it is a leaf incident to an explored edge.

A way of exploring exactly one edge in a phase is as follows. One of the agents walks through the explored territory to a limit p, while the other agent walks through the explored territory to p, traverses an unknown edge and returns to p. If we assume that both agents are at a limit p of the explored territory at step t and (p,u) is an unknown edge towards node v, we define the following procedure.

Probe(v). One agent traverses edge (p,u) (which is towards node v) and returns to node p to meet the other agent who waits. If they do not meet at step $t+2$ then the black hole has been found.

We also define a procedure that the two agents could follow to explore two new edges in a phase. Suppose that the two agents reside at node m at step t. Let $p_1,\ldots,p_i$ be the limits of the explored territory at that step. Each of the unknown edges which could be explored in the following phase has to be incident to a node from the set $\{p_1,\ldots,p_i\}$. Let

the two selected unknown edges for exploration be (p_k, k) and (p_l, l), $p_k, p_l \in \{p_1, \ldots, p_i\}$ (possibly $p_k = p_l$). We assume that node m belongs to the path $< k, p_k, \ldots, p_l, l >$. The definition of the procedure is as follows.

Split(k, l). One of the agents traverses the path from node m to node k and returns towards node p_l. The other traverses the path from node m to node l and returns towards node p_k. Let $\mathrm{dist}(l,k)$ denote the number of edges in the path from node k to node l. If they do not meet at step $t + \mathrm{dist}(l,k)$ then the black hole has been found.

4. Black hole search in a line

In this section we construct an optimal black hole search algorithm for lines, with linear time complexity. A line is a graph $L = (V, E)$, where $V = \{0, \ldots, n\}$ and $E = \{(i, i+1) : i = 0, 1, \ldots, n-1\}$. 0 and n are called endpoints of the line. The starting node is denoted by s, while a and b denote the distances between s and the endpoints of the line, with $a \leqslant b$, hence $a + b = n$. We assume that L is represented by a horizontal line segment in the plane, with a being the distance from s to the rightmost vertex of L and b being the distance from s to the leftmost vertex of L. Hence the direction right of the starting node means that of the closer endpoint. We assume $b > 0$, otherwise the line consists of a single node.

Lemma 4.1. *Consider a BHS-scheme A on L. Suppose that meetings occur in A at steps t_i at nodes p_i and consider the first meeting that occurs in A at step t_1 such that the explored territory after the meeting is* Expl *with* $|\mathrm{Expl}| \geqslant 2$. *Then we can transform A into a BHS-scheme A' so that:*

- *for every meeting which occurs in A' such that the explored territory* Expl$'$ *after the meeting has size* $|\mathrm{Expl}'| \geqslant 2$ *it holds that the meeting point is not a limit of* Expl$'$, *and*
- *at any step $t_i \geqslant t_1$ of A/A', the agents in both BHS-schemes are at the same node and the explored territory is the same.*

Proof. Suppose that during the execution of A a meeting occurs at step t_i at a node p_i which is a limit of the explored territory Expl at step t_i.

If p_i is not an endpoint of an edge which was already explored at step $t_i - 1$ (*i.e.*, both edges incident to p_i were unexplored at step $t_i - 1$), then by Lemma 3.4 we have $p_i = s$. The explored territory cannot be disconnected, hence at step $t_i - 1$ the set Expl was empty. Since $|\mathrm{Expl}| \geqslant 2$ at step t_i, the two agents traversed two edges on different sides of p_i. Hence, in this case, p_i could not be a limit of territory Expl at step t_i.

Suppose that p_i is an endpoint of an edge which was already explored at step $t_i - 1$. By Lemmas 3.1 and 3.2, exactly one agent R_1 did not traverse an unexplored edge during the previous phase (otherwise p_i could not be a limit of the explored territory at step t_i). This means that agent R_2 has traversed an unexplored edge at the other limit of the explored territory and returns to node p_i.

At step $t_i - 1$ agent R_2 has to be at a node p_i' which is adjacent to p_i. At step $t_i - 1$, agent R_1 has to be already at node p_i because otherwise, the meeting would have to take

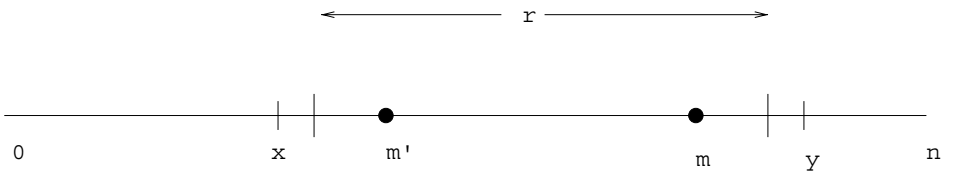

Figure 1. An exploration of two new edges in a phase needs at least $r+2$ time units.

place at step $t_i - 1$. Thus at step $t_i - 2$, agent R_1 is either at node p_i or p_i'. We transform the BHS-scheme A as follows:

- if R_1 is at node p_i' at step $t_i - 2$: R_1 waits for one step,
- if R_1 is at node p_i at step $t_i - 2$: R_1 walks to node p_i',
- R_1 walks to node p_i.

Observe the following. The meeting in A' takes place at step $t_i - 1$ at node p_i', which is not a limit of the explored territory at step $t_i - 1$. At step t_i the two agents are at the same node p_i and the explored territory is the same. □

Hence, for any BHS-scheme A which explores the line in time t, there is a BHS-scheme A' which explores the line in the same time t and no meeting point in A' is a limit of the explored territory after the meeting (assuming that the explored territory includes at least two edges).

A BHS-scheme on L with the property of Lemma 4.1 is called *regular*.

Lemma 4.2. *Consider a regular BHS-scheme A. Let $r = |\,\mathrm{Expl}\,|$ be the size of the explored territory and m be the meeting point at the end of a phase in A. Let p, p' be the limits of* Expl *and $d = \min\{|m-p|, |m-p'|\}$. If the next phase in A is a 1-phase then the time needed for it is at least 2 units if $r \leqslant 1$ and $d+2 \geqslant 3$ units if $r \geqslant 2$.*

Proof. At any 1-phase, an agent has to walk distance d in order to reach the limit of the explored territory and then has to traverse an unknown edge and return. Hence at least $d+2$ time units are needed. If $r \geqslant 2$ then regularity of A implies $d \geqslant 1$ (m cannot be a limit of Expl), which concludes the lemma. □

Lemma 4.3. *Consider a BHS-scheme A. Let $r = |\,\mathrm{Expl}\,|$ be the size of the explored territory and m be the meeting point at the end of a phase in A. If the next phase in A is a 2-phase then the time needed for it is at least $r+2$ units. Furthermore, if the time is exactly $r+2$, then the new meeting point is $m' = x + y - m$, where $x+1$ and $y-1$ are, respectively, the left and right limit of the explored territory before the phase* (*see Figure 1*).

Proof. In view of Lemmas 3.1 and 3.2, the two edges which will be explored in the 2-phase must be right and left of m. The time spent by the agent which goes right of m until the meeting at m' is at least $2y - m - m'$. At the same time the other agent travelled distance $m + m' - 2x$. Thus, $m + m' = x + y$ and the time spent is at least $r+2$. The second part of the lemma is straightforward. □

Lemmas 4.2 and 4.3 imply the following.

Lemma 4.4. *Suppose that the explored territory consists of at least* 4 *edges at the end of a phase. Then, in any regular BHS-scheme, the two agents need at least* 3 *time units to explore exactly one additional edge and at least* 6 *time units to explore two additional edges* (*in two 1-phases or one 2-phase*).

Lemma 4.5. *Suppose* $a \geqslant 1$. *Any regular BHS-scheme A whose first phase is a 1-phase can be transformed into a regular BHS-scheme* A' *whose first phase is a 2-phase and* $\text{time}(A') \leqslant \text{time}(A)$.

Proof. Consider a regular BHS-scheme A. Suppose w.l.o.g. that A starts with a 1-phase left of the starting node. We transform A into a regular BHS-scheme A' as follows.

A' starts executing procedure split$(s-1, s+1)$; after 2 time units in both BHS-schemes the two agents are in s. The BHS-scheme A' has explored one additional edge to the right of s.

- Suppose that A continues with a 1-phase exploring one edge right of s and let m be the meeting point at the end of that phase. By regularity of A and since the size of the explored territory is at least 2, we have $m = s$. In A' both agents wait for 2 time units and after that the situation in both BHS-schemes is exactly the same. Then A' continues by just copying A. If a black hole is discovered by the BHS-scheme A during the first two phases, then it must have been discovered by the BHS-scheme A' in the first phase. After the first two phases of A, the BHS-schemes are identical. Hence $\text{time}(A') \leqslant \text{time}(A)$.
- Otherwise, A' copies A until a phase ϕ in which A explores the first edge right of the starting node (this edge is already explored by A'). Suppose that ϕ lasts t steps. Let m be the meeting point after this phase. If ϕ was a 1-phase then both agents in A' walk to m (which can be done in less than t steps). If ϕ was a 2-phase, let p be the left limit of the explored territory before ϕ and let (p, v) be the edge left of s, which was explored in ϕ. Then the agents in A' walk to p, execute probe(v) and then walk to m (this can be done in at most t steps). Again, if a black hole is discovered by scheme A in a step before phase ϕ, then it must have been discovered by scheme A' by that step. After phase ϕ the two BHS-schemes are identical. Therefore $\text{time}(A') \leqslant \text{time}(A)$. □

In view of Lemma 4.5 we may restrict attention to regular BHS-schemes which start with a 2-phase.

The following lemma and the next two theorems give lower bounds on the exploration time of the line by any regular BHS-scheme. The idea of their proofs is as follows. Take a regular BHS-scheme A and consider the first meeting at step t_1 at node m when the explored territory Expl of size $r = |\,\text{Expl}\,|$ is such that one of the two limits p, p' of Expl is adjacent to one endpoint of the line (see Figure 2). Also, let t_2 be the time needed by A to explore the remaining territory which has size $n - r$. In Lemma 4.7 the lower bounds for time t_2 are given, while in Theorems 4.8 and 4.9 we provide the lower bounds for time t_1 and compute the total time needed by the scheme A for black hole search.

Lemma 4.6. *Let A be a regular BHS-scheme. Suppose that exactly two edges on different sides of a meeting point are explored by A in two consecutive* 1-phases. *We can transform the*

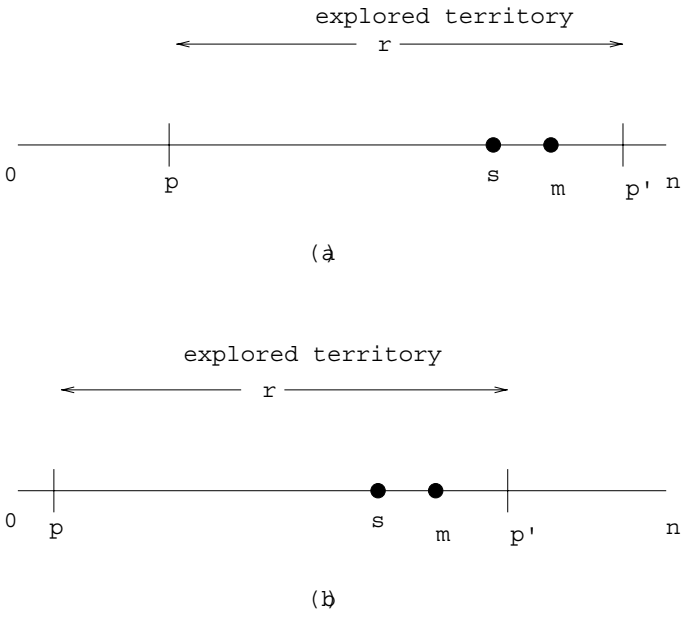

Figure 2. Exploration of the extremal edges.

BHS-scheme A into a BHS-scheme A′ so that these two edges are explored in one 2-phase and $\text{time}(A') \leqslant \text{time}(A)$.

Proof. Let $r = |\,\text{Expl}\,|$ be the size of the explored territory, m the meeting point, and $x+1$, $y-1$ the two limits of the explored territory at the end of a phase as in Figure 1. Suppose, without loss of generality, that $y - m \leqslant m - x$. The time needed for exploring two new edges right and left of m in the next two *1-phases* is at least $y - m + 1 + r + 2 = r + y - m + 3$, while the new meeting point is at $x + 1$.

Exploring them by executing procedure split(x, y) (which is a 2-phase) uses $r + 2$ time units and the new meeting point is at $x + y - m$. It requires $x + y - m - x - 1 = y - m - 1$ additional time units to walk to $x + 1$. So the total time is $r + y - m + 1$. □

Lemma 4.7. *Let A be a regular BHS-scheme. Let* $a \geqslant 3$. *Consider the first meeting at step* t_1, *when the explored territory* Expl *of size* $r = |\,\text{Expl}\,|$ *is such that one of the two limits p, p′ of* Expl *is adjacent to one endpoint of the line. Let m be the position of this meeting point* (*see Figure 2*). *Then the time* t_2 *needed by scheme A to explore the remaining territory of size* $n - r$, *is at least as follows.*

- $r + 2 + |n - m - s|$ *time units, when* $n - r = 2$.
- *When* $n - r > 2$ *and territory* Expl *is adjacent to the right endpoint:*
  - $3n - 2r + b - 4$ *time units, when the previous phase* (*before step* t_1) *is a 1-phase,*
  - $3n - 2r + a + m - 8$ *time units, when the previous phase* (*before step* t_1) *is a 2-phase and* $m \leqslant b + 1$,

- $4n - 2r - m + b - 6$ *time units, when the previous phase (before step* t_1*) is a 2-phase and* $m > b + 1$.

- *When* $n - r > 2$ *and territory* Expl *is adjacent to the left endpoint:*
  - $3n - 2r + a - 4$ *time units, when the previous phase (before step* t_1*) is a 1-phase,*
  - $4n - 2r + b - m - 8$ *time units, when the previous phase (before step* t_1*) is a 2-phase and* $m \geqslant b - 1$,
  - $3n - 2r + m + a - 6$ *time units, when the previous phase (before step* t_1*) is a 2-phase and* $m < b - 1$.

Proof. If $n - r = 2$, then by Lemma 4.3 $t_2 \geqslant r + 2 + |n - m - s|$.

Suppose that $n - r > 2$. By Lemma 4.6, exploring two edges on both sides of s by a 2-phase requires less or equal time than exploring them by two consecutive 1-phases. Thus the minimum time is achieved when the remaining edge is explored together with an edge in the opposite direction by a 2-phase. Furthermore, the minimum time is achieved when this 2-phase is the first phase after step t_1 or the last phase.

Suppose that territory Expl is adjacent to the right endpoint (see Figure 2(a)). Note that in this case, Lemma 4.5 implies $r \geqslant a$. Using Lemmas 4.2 and 4.3 we compute the following times.

- If the previous phase (before step t_1) is a 1-phase then $m = n - 2$.
  - If the first phase (after step t_1) is a 2-phase then $t_2 \geqslant r + 3 + 3(n - r - 2) + b - 1$.
  - If the last phase is a 2-phase then $t_2 \geqslant r - 2 + 3(n - r - 2) + \max\{b + 2, b + 2a - 2\}$.

  In this case it is easy to see that when the first phase (after step t_1) is a 2-phase then time t_2 is minimum.
- If the previous phase (before step t_1) is a 2-phase then:
  - If the first phase (after step t_1) is a 2-phase then $t_2 \geqslant n - m + r + 1 + 3(n - r - 2) + b - 1$.
  - If the last phase is a 2-phase then $t_2 \geqslant m - n + r + 3(n - r - 2) + \max\{b + 2, b + 2a - 2\}$.

Suppose that territory Expl is adjacent to the left endpoint (see Figure 2(b)). Note that in this case Lemma 4.5 implies $r \geqslant b$.

- If the previous phase (before step t_1) is a 1-phase then $m = 2$.
  - If the first phase (after step t_1) is a 2-phase then $t_2 \geqslant r + 3 + 3(n - r - 2) + a - 1$.
  - If the last phase is a 2-phase then $t_2 \geqslant r - 2 + 3(n - r - 2) + \max\{a + 2, a + 2b - 2\}$.

  In this case it is easy to see that when the first phase (after step t_1) is a 2-phase then time t_2 is minimum.
- If the previous phase is a 2-phase then:
  - If the first phase (after step t_1) is a 2-phase then $t_2 \geqslant m + r + 1 + 3(n - r - 2) + a - 1$.
  - If the last phase is a 2-phase then $t_2 \geqslant r - m + 3(n - r - 2) + \max\{b + 2, a + 2b - 2\}$. □

Theorem 4.8. *The time of any regular BHS-scheme on the line is at least* $4n-8$ *when* $a \geqslant 6$.

Proof. Let r, t_1, t_2 be defined as above. In view of Lemma 4.5 we assume that the first phase is a 2-phase, hence $r \geqslant a$. Depending on the value of r as well as to which endpoint Expl at step t_1 is adjacent, several cases are possible.

If $n-r=2$ it means that the explored territory is adjacent to both endpoints of the line. Hence the last phase before step t_1 must be a 2-phase (otherwise the first meeting with Expl adjacent to an endpoint would have occurred before step t_1) and in view of Lemma 4.4, $t_1 \geqslant 2+4+3(n-8)+n-2$. According to Lemma 4.3 we have $t_2 \geqslant n$, thus the total time needed is at least $5n-20$.

If $n-r>2$ and territory Expl is adjacent to the right endpoint then:

- If $r=a$, since the first phase was a 2-phase, the last phase before step t_1 must be a 1-phase and $t_1 \geqslant 2+3(a-2)$. Lemma 4.7 implies $t_2 \geqslant 3n-2a+b-4$. Hence the total time is at least $4n-8$.
- If $r=a+1$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+1+3(a-3)$. Lemma 4.7 implies $t_2 \geqslant 3n-2a-2+b-4$. Hence the total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then Lemma 4.3 implies $m=n-a$ and $t_1 \geqslant 2+3(a-3)+a+1$. Lemma 4.7 implies $t_2 \geqslant 3n-2a-2+a+n-a-8$. Hence the total time is at least $4n-8$.
- If $r \geqslant a+2$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+3(r-4)$. Lemma 4.7 implies $t_2 \geqslant 3n-2r+b-4$. The total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then $t_1 \geqslant 2+4+3(r-6)+r$. The coordinates of the meeting point m' before that last phase have to be $n-r+1 \leqslant m' \leqslant n-3$ (in view of Lemma 4.1). Lemma 4.3 implies that the coordinates of meeting point m have to be $n-r+1 \leqslant m \leqslant n-3$. In view of Lemma 4.7, t_2 is either at least $3n-2r+a+m-8 \geqslant 4n-3r+a-7$ or at least $4n-2r-m+b-6 \geqslant 3n-2r+b-3$. In both cases the total time is greater than $4n-8$.

If $n-r>2$ and territory Expl is adjacent to the left endpoint, the total time is at least $4n-8$, using a similar argument.

Since $4n-8 \leqslant 5n-20$ when $a \geqslant 6$, this proves the theorem. □

The following theorem establishes lower bounds for the remaining cases.

Theorem 4.9. *The time of any regular BHS-scheme A on the line is at least:*

- $4n-2$, *when* $a=0$,
- $\sum_{i=1}^{a} 2i$, *when* $1 \leqslant a=b \leqslant 5$,
- $4n-6$, *when* $a=1<b$,
- $4n-10$, *when* $a=2<b$ *or* $a=3<b$,
- $4n-8$, *when* $a=4<b$ *or* $a=5<b$.

Proof. **(case $a = 0$)** In view of Lemmas 3.1 and 3.2, the BHS-scheme A explores 1 new edge in every phase. Thus, by Lemma 4.2, the total time is at least $2 + 4(n-1)$.

(case $a = 1$) In view of Lemma 4.5 we may assume that the first phase is a 2-phase. After that only 1-phases are possible. So if $a = b = 1$ the time is at least 2. If $a < b$ the total time is at least $2 + 3(b-1) + b - 1 = 4n - 6$.

(case $a = 2$) In view of Lemma 4.5 we may assume that the first phase is a 2-phase. If $a = b = 2$ then, by Lemma 4.6, we may assume that the exploration of the two remaining edges left and right of s is done in a 2-phase and the total time is at least 6. If $a < b$ then:

- If the second phase is a 2-phase then the total time is at least $2 + 5 + 3(b-2) + b - 1 = 4n - 8$.
- If the last phase is a 2-phase then the total time is at least $2 + 3(b-2) + b + 2 = 4n - 10$.

(case $a = 3$) As before, we may assume that the first phase is a 2-phase. Consider the first meeting at step t_1, when the explored territory Expl with size $r = |\,\text{Expl}\,|$ is such that one of the two limits p, p' of Expl is adjacent to one endpoint of the line. Let m be the meeting point at t_1 and let t_2 be the time needed by A to explore the remaining territory of size $n - r$.

If $n - r = 2$ then the explored territory is adjacent to both endpoints of the line. Hence the last phase before step t_1 must be a 2-phase (otherwise the first meeting with Expl adjacent to an endpoint would have occurred before step t_1). Hence $t_1 \geqslant 2 + 3(b-3) + n - 2$ and, by Lemma 4.3, we have $m = n - 3$. By Lemma 4.7, $t_2 \geqslant n + |n - n + 3 - n + 3|$. Thus the total time is at least $5n - 18 + |n - 6|$. In particular, if $a = b = 3$ then the time is at least 12 units. If $a < b$ then the time is at least $6n - 24 \geqslant 4n - 10$.

If $n - r > 2$ then we have:

- If territory Expl is adjacent to the right endpoint then:
  - If $r = a$, since the first phase is a 1-phase, the last phase before step t_1 must be a 1-phase right of s and $t_1 \geqslant 5$. By Lemma 4.7, $t_2 \geqslant 3n - 2a + b - 4$. Hence the total time is at least $4n - 8$.
  - If $r \geqslant a + 1$ then let r' be the size of the territory left of s which has been explored by step t_1. Lemma 4.6 implies that the last two edges which were explored before step t_1, must have been explored by a 2-phase. Thus, by Lemma 4.3, $m = n - 3$ and $t_1 \geqslant 2 + 3(r' - 2) + 2 + r'$. By Lemma 4.7, $t_2 \geqslant 3n - 2r' - 4 + 3 + n - 3 - 8$. Hence the total time is at least $4n - 10$.
- If territory Expl is adjacent to the left endpoint, then the last phase before step t_1 must be a 1-phase. In view of Lemma 4.2 we have $t_1 \geqslant 2 + 3(b-2)$ and in view of Lemma 4.7 we have $t_2 \geqslant 3n - 2b + a - 4$. Hence the total time is at least $4n - 8$.

(case $a = 4$ or $a = 5$) If $n - r = 2$ then the explored territory is adjacent to both endpoints of the line. Hence the last phase before step t_1 must be a 2-phase (otherwise the first meeting with Expl adjacent to an endpoint would have occurred before step t_1) and in view of Lemma 4.4 we have $t_1 \geqslant 2 + 4 + 3(n-8) + n - 2$. In view of Lemma 4.3 we have $t_2 \geqslant n$, thus the total time needed is at least $5n - 20$. In particular, the total time is at least 20 when $a = 4$ and 30 when $a = 5$. However, if $a < b$, then $m = n - 3$. Thus

$t_1 \geqslant 2+4+1+3(b-4)+n-2$ when $a=4$ and $t_1 \geqslant 2+4+6+2+3(b-5)+n-2$ when $a=5$. In view of Lemma 4.7 we have $t_2 \geqslant n+|n-n+3-n+4|$. Thus the total time is at least $6n-26$ when $a=4$ and at least $6n-25$ when $a=5$.

If $n-r>2$ and territory Expl is adjacent to the right endpoint then:

- If $r=a$, since the first phase was a 2-phase, the last phase before step t_1 must be a 1-phase and $t_1 \geqslant 2+3(a-2)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2a+b-4$. Hence the total time is at least $4n-8$.
- If $r=a+1$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+1+3(a-3)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2a-2+b-4$. Hence the total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then, in view of Lemma 4.3, we have $m=n-a$ and $t_1 \geqslant 2+3(a-3)+a+1$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2a-2+a+n-a-8$. Hence the total time is at least $4n-8$.
- If $r \geqslant a+2$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+3(r-4)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2r+b-4$. The total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then $t_1 \geqslant 2+4+3(r-6)+r$. If m' is the meeting point before this last phase then $n-r+1 \leqslant m' \leqslant n-3$. In view of Lemma 4.3 we have $n-r+1 \leqslant m \leqslant n-3$. In view of Lemma 4.7, t_2 is either at least $3n-2r+a+m-8 \geqslant 4n-3r+a-7$ or at least $4n-2r-m+b-6 \geqslant 3n-2r+b-3$. In both cases the total time is greater than $4n-8$.

If $n-r>2$ and territory Expl is adjacent to the left endpoint then:

- If $r=b$, then, since the first phase is a 2-phase, the last phase before step t_1 must be a 1-phase and $t_1 \geqslant 2+3(b-2)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2b+a-4$. Hence the total time is at least $4n-8$.
- If $r=b+1$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+1+3(b-3)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2b-2+a-4$. Hence the total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then Lemma 4.3 implies $m=n-a$ and $t_1 \geqslant 2+3(b-3)+b+1$. In view of Lemma 4.7 we have $t_2 \geqslant 4n-2b-2+b-n+a-8$. Hence the total time is at least $4n-8$.
- If $r \geqslant b+2$ then:
  - If the last phase before step t_1 is a 1-phase then $t_1 \geqslant 2+4+3(r-4)$. In view of Lemma 4.7 we have $t_2 \geqslant 3n-2r+a-4$. The total time is at least $4n-8$.
  - If the last phase before step t_1 is a 2-phase then $t_1 \geqslant 2+4+3(r-6)+r$. If m' is the meeting point before this last phase then $3 \leqslant m' \leqslant r-1$. In view of Lemma 4.3 we have $3 \leqslant m \leqslant r-1$. In view of Lemma 4.7, t_2 is either at least $3n-2r+m+a-6 \geqslant 3n-2r+a-3$ or at least $4n-2r+b-m-8 \geqslant 4n-3r+b-7$. In both cases the total time is greater than $4n-8$.

Since $4n-8 \leqslant 6n-26$ when $n \geqslant 9$, this proves the theorem. □

We will now give an optimal algorithm to solve the black hole search problem for the line (*i.e.*, an algorithm which produces a fastest BHS-scheme for any line). The algorithm uses procedures *probe*, *split* and the following ones:

- **walk(k)**: both agents go 1 step towards node k.
- **walk-and-probe(v)**:
  while the position of the agents is not adjacent to node v **do**
    walk(v);
    probe(v)
- **return(s)**:
  repeat walk(s) **until** all remaining agents are at s

Suppose that both agents reside at the same node m. The high-level description of Algorithm *Line* is as follows.

- **Case** $a = 0$**:** the two agents explore the line by probing left of s and return.
- **Case** $1 \leqslant a = b \leqslant 5$**:** the two agents explore the line by repeated splits.
- **Case** $a = 1 < b$**:** the two agents first do a split and then explore the rest of the line by probing left and return.
- **Case** $a = 2 < b$**:** the two agents first do a split, then explore all edges left of s except one by probing, and finally explore the last two edges by a split.
- **Case** $3 \leqslant a < b$ **or** $a \geqslant 6$**:** the two agents first do two splits, then explore all edges left of s except one by probing. They explore the last left edge together with an edge right of s by a split and finally explore the remaining edges (if any) which are right of s by probing and return.

The precise formulation of Algorithm Line is shown opposite. The time complexity of the algorithm is clearly linear.

Notice that after any probe, split or walk-and-probe phase, at least one agent is alive. During each walk procedure, the agents are always in the explored territory. At the end of the algorithm all edges are explored. Therefore the following lemma holds.

Lemma 4.10. *Algorithm Line produces a BHS-scheme for any line.*

Lemma 4.11. *The time of the BHS-scheme produced by Algorithm Line is at most:*

- $4n - 2$, *when* $a = 0$,
- $\sum_{i=1}^{a} 2i$, *when* $1 \leqslant a = b \leqslant 5$,
- $4n - 6$, *when* $a = 1 < b$,
- $4n - 10$, *when* $a = 2 < b$ *or* $a = 3 < b$,
- $4n - 8$, *when* $a = 4 < b$ *or* $a = 5 < b$ *or* $a \geqslant 6$.

Proof. Procedure probe requires two time units. Procedure split(k, l) requires $|l - k|$ time units.

If $a = 0$ then procedure walk-and-probe requires $2 + 3(n - 1)$ time units while $n - 1$ time units are required for returning to s. Hence the total time is $4n - 2$.

If $1 \leqslant a = b \leqslant 5$, then the total time is $\sum_{i=1}^{a} 2i$.

Algorithm Line.

case $a = 0$
    probe(0);
    walk-and-probe(0);
case $1 \leqslant a = b \leqslant 5$
    for $i := 1$ **to** a
        split($s - i$, $s + i$);
case $a = 1 < b$
    split($s - 1$, $s + 1$);
    walk-and-probe(0);
case $a = 2 < b$
    split($s - 1$, $s + 1$);
    walk-and-probe(1);
    split(0, $s + 2$)
case $a = 3 < b$
    split($s - 1$, $s + 1$);
    split($s - 2$, $s + 2$);
    walk($s - 1$);
    walk-and-probe(1);
    split(0, $s + 3$);
case $4 \leqslant a < b$ **OR** $a \geqslant 6$
    split($s - 1$, $s + 1$);
    split($s - 2$, $s + 2$);
    walk($s - 1$);
    walk-and-probe(1);
    split(0, $s + 3$);
    walk($s + 2$);
    walk-and-probe(n);
return(s)

If $a = 1 < b$ then the time for the split is 2. The meeting point after the split is at s, while the limit of the explored territory is at $s - 1$. Thus walk-and-probe takes time $3(n - 2)$ and returning takes time $n - 2$, for a total time $4n - 6$.

If $a = 2 < b$ then the time for the split is 2. The meeting point after the split is at s, while the limit of the explored territory is at $s - 1$. Thus walk-and-probe takes time $3(n - 4)$. The last split takes time n, while the last meeting point is at s. Hence the total time is $4n - 10$.

For the remaining cases the time for the two first splits is $2 + 4$ and the meeting point is at s.

- If $a = 3$ then walk takes 1 time unit and after that the two agents are at $s - 1$ while the limit of the explored territory is at $s - 2$. Thus walk-and-probe takes time $3(n - 6)$.

The last split takes time n and after that the two agents are at $s+1$. So returning to s takes another 1 time unit. The total time is $7+3(n-6)+n+1=4n-10$.

- If $4 \leqslant a \leqslant 5$ and $a < b$, or $a \geqslant 6$ then walk takes 1 time unit and after that the two agents are at $s-1$ while the limit of the explored territory is at $s-2$. Thus walk-and-probe takes time $3(b-3)$ and the meeting point is at 2. The next split takes $b+3$ time units and the meeting point is at $s+1$. The next walk takes 1 time unit and after that the two agents are at $s+2$ while the limit of the explored territory is at $s+3$. Thus walk-and-probe takes time $3(a-3)$. The final meeting point is at $n-1$, so returning to s takes $a-1$ additional time units. The total time is $7+3(b-3)+b+3+1+3(a-3)+a-1=4n-8$. □

Theorems 4.9 and 4.8 and Lemmas 4.10 and 4.11 imply the following result.

Theorem 4.12. *Algorithm Line produces a fastest BHS-scheme for any line.*

5. Black hole search in a tree

In this section we study the problem of black hole search in trees.

Consider a tree T rooted at the starting node s. If e is an edge, $e=(u,v)$ means that v is the child of u. Let $e=(u,v)$ be an edge of the tree. Consider the following colouring, which creates a partition of the edges of the tree. This partition will be used in the analysis of our algorithms.

- Assign red colour to edge e if node v has at least two descendants.
- Assign green colour to edge e if v is a leaf and exactly one of the following holds: $u=s$ or the edge (t,u) is a red edge (where t is the parent of u).
- Assign blue colour to edge e if it has none of the above properties.

Red, green and blue edges are shown in Figure 3.

Let $e=(u,v)$ and $e'=(v,z)$ be two blue edges as shown in Figure 3(c), *i.e.*, v is a child of u and z is a leaf and the unique child of v. We call the set of these two edges a *branch*. The set of all branches of blue edges with upper node u is called a *block*.

Let u be a node with at least two children. Let v_i, v_j be two children of u. We call the edges $e_i=(u,v_i)$, $e_j=(u,v_j)$ sibling edges.

Lemma 5.1. *In any BHS-scheme, the following holds: a green edge has to be traversed by the agents at least* 2 *times, a red edge has to be traversed at least* 6 *times, and a branch of blue edges requires a total of at least* 6 *traversals.*

Proof. By Lemma 3.1 any edge has to be traversed 2 times by one agent to become explored. In particular a green edge needs 2 traversals.

Consider a red edge $e=(u,v)$. Let l be the number of descendants of node v. In view of Lemmas 3.1 and 3.2, if, during any phase after its exploration, edge e is traversed always by only one agent then at least $2l \geqslant 4$ additional traversals are required (an agent has to traverse e twice for every descendant of v). If there is at least one phase after exploration

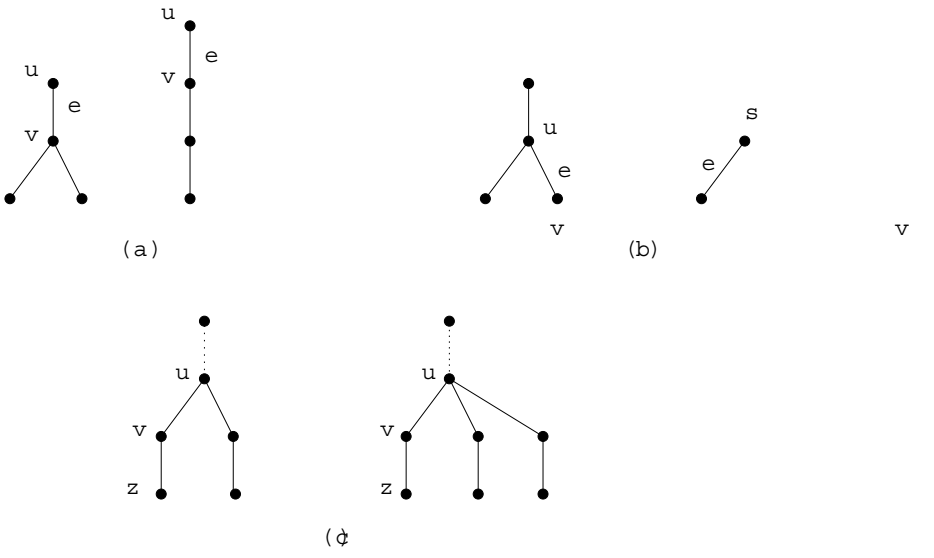

Figure 3. In (a) e is a red edge. In (b) e is a green edge. In (c) all solid edges are blue.

of e where the edge is traversed by both agents then at least 4 additional traversals of e are required for the exploration of the edges with upper node v (both agents traverse e and return). Thus the total minimum number of traversals is 6.

A branch of 2 blue edges can be traversed in the following ways. Two traversals are required for the exploration of the upper edge of the branch. If, during any phase after exploration of the upper edge, this edge is traversed always by only one agent then at least 4 additional edge traversals on this branch are required. If there is at least one phase after exploration of the upper edge when this edge is traversed by both agents then at least 6 additional edge traversals on this branch are required (both agents traverse the upper edge, then one of them explores the lower edge and finally they return). Therefore the total minimum number of traversals on each branch is 6. □

Since, in any BHS-scheme, each of the two agents can do at most one traversal in every time unit, the following lemma holds.

Lemma 5.2. *In any BHS-scheme the two agents need at least* $\lceil \frac{l}{2} \rceil$ *time units to perform* l *traversals.*

Lemmas 5.1 and 5.2 imply the following.

Lemma 5.3. *Any BHS-scheme requires at least* 3, 1 *and* 3b *time units for the traversals of a red edge, a green edge and a block of* b *branches of blue edges, respectively.*

5.1. An optimal algorithm for a family of trees

Consider the family $\mathcal{T}$ of rooted trees with the following property: any internal node of a tree in $\mathcal{T}$ (including the root) has at least 2 children. Trees in $\mathcal{T}$ will be called *bushy trees.*

Algorithm Bushy-Tree.
special-explore(s)

Procedure special-explore(v).
for every pair of unknown edges $(v,x),(v,y)$ with upper node v **do**
split(x,y), so that edge $(v,L(v))$ is explored last
end for
if every edge is explored **then**
repeat walk(s) **until** (all remaining agents are at s)
else
case 1: every edge incident to v has been explored
next := relocate(v);
special-explore(*next*);
case 2: there is an unknown edge (v,z) incident to v
(* *must be* $z = L(v)$ *)
explore-only-child(v, *next*);
special-explore(*next*);
end if

For these trees, searching for a black hole can be efficiently parallelized by an appropriate use of the procedure split. This enables us to get an optimal algorithm for this class of trees.

Let T be a bushy tree with root s and let u be an internal node of T. The *heaviest child* $v = H(u)$ of u is defined as a child v of u such that the subtree $T(v)$ rooted at v (which is also a bushy tree) has a maximum height among all subtrees rooted at children of u. The *lightest* child $v' = L(u)$ of u is defined as a child v' of u such that the subtree $T(v')$ rooted at v' has a minimum height among all subtrees rooted in a child of u. Ties are broken arbitrarily. Notice that $H(u)$ and $L(u)$ can be computed for all nodes u in linear time.

The high-level description of Algorithm *Bushy-Tree* is as follows. Let m be the meeting point of the two agents after a phase (initially $m = s$).

- Explore any pair of unknown edges (m,x), (m,y) with upper node m by executing procedure split(x,y), leaving edge $(m,L(m))$ last.
- If there is one unknown edge with upper node m (which must be $(m,L(m))$), explore this edge together with another unknown edge (if any), again using procedure split. If edge $(m,L(m))$ is the last unknown edge in the tree, explore it by executing procedure probe($L(m)$).
- If all edges with upper node m are explored, explore, as before, any unknown edges incident to the children of m and to ancestors of m.

Function *relocate*(v) takes as input the current node v where both agents reside and returns the new location of the two agents. If there is an unknown edge incident to a child of v then the agents go to that child. Otherwise the two agents go to the parent of v.

Function relocate(v).

case 1.1: $\exists$ an unknown edge incident to $w \in children(v)$
walk(w);
relocate := w

case 1.2: every edge incident to any child of v is explored
let t be the parent of v;
walk(t);
relocate := t

Procedure explore-only-child($v, next$).

case 2.1: there is an unknown edge incident to $w \in children(v)$, $w \neq L(v)$
split($L(v)$,$H(w)$);
$next := w$

case 2.2: every edge incident to any $w \in children(v)$, $w \neq L(v)$ is explored
(* $L(v)$ *must be a leaf* *)

case 2.2.1: there are at least 2 unknown edges left
let a be the deepest ancestor of v having a descendant
incident to an unknown edge (excluding $L(v)$);
$D(a)$:= the closest descendant of a with incident unknown edges;
split($H(D(a))$, $L(v)$);
$next := D(a)$

case 2.2.2: there is only 1 unknown edge left
probe($L(v)$);
$next := v$

Procedure *explore-only-child*$(v, next)$ takes as input the current node v where both agents reside and returns the new meeting point after the exploration of edge $(v, L(v))$. The description of the procedure is as follows.

- If there is an unknown edge incident to a child w of v, $w \neq L(v)$, then the agents explore edge $(w, H(w))$ together with edge $(v, L(v))$ by split$(H(w), L(v))$. The new meeting point is w.
- If every edge incident to any child w of v, different from $L(v)$, is explored and edge $(v, L(v))$ is not the last unknown edge in the tree, then find the deepest ancestor a of v having a descendant incident to an unknown edge (excluding $L(v)$); the agents explore edge $(D(a), H(D(a)))$ (where $D(a)$ is the closest descendant of a with incident unknown edges), together with edge $(v, L(v))$, by split$(H(D(a)), L(v))$; the new meeting point is $D(a)$.
- If edge $(v, L(v))$ is the last unknown edge in the tree then explore it by calling probe($L(v)$); the new meeting point is v.

Notice that all edges of the tree (except possibly the last one if the number of edges is odd) are explored by calling procedure split. Observe that in any bushy tree, there are

only *red* and *green* edges. By definition, in every red edge $e_r = (u_r, v_r)$, node v_r has at least two children and every leaf of the tree is an endpoint of a green edge $e_g = (u_g, v_g)$. Also u_g has at least two children.

Lemma 5.4. *Algorithm Bushy-Tree produces a BHS-scheme for any bushy tree.*

Proof. Consider a bushy tree T. We prove that Algorithm Bushy-Tree explores all the edges of T.

- If every internal node of the tree has an even number of children, then the two agents explore all edges by calling procedure split (case 1 in procedure special-explore).
- Otherwise, consider the first meeting point v incident to an odd number of unknown edges. The two agents explore all edges with upper node v except the edge $(v, L(v))$ by calling split.

  If $L(v)$ is a leaf ($(v, L(v))$ is a green edge).
  - If this is the last unknown edge in the tree then the two agents explore it by calling procedure probe (case 2.2.2 in procedure explore-only-child).
  - Otherwise (*i.e.*, there is at least one more unknown edge e) the agents explore the edge $(v, L(v))$ together with edge e by calling split:
    - either edge e is incident to a child w of v (case 2.1 in procedure explore-only-child), or
    - edge e is incident to a descendant of an ancestor of v (case 2.2.1 in procedure explore-only-child)

  If $L(v)$ is an internal node ($(v, L(v))$ is a red edge) then edge $(v, L(v))$ cannot be the last unknown edge in the tree. There must also be another child w of v with an unknown edge (w, z) incident to w because $L(v)$ is a lightest child of v. Hence edge $(v, L(v))$ is explored together with edge (w, z) by calling split$(z, L(v))$ (case 2.1 in procedure explore-only-child). □

Since all values $H(u)$ and $L(u)$ can be computed in linear time it is easy to see that the time complexity of Algorithm Bushy-Tree is linear.

Theorem 5.5. *Algorithm Bushy-Tree produces a fastest BHS-scheme for any bushy tree.*

Proof. Consider a bushy tree T. We prove that the scheme produced by Algorithm Bushy-Tree traverses any red edge 6 times and any green edge 2 times. Moreover every phase is a 2-phase (*i.e.*, the two agents traverse edges in parallel), except possibly the last phase, and no agent waits in any 2-phase.

Notice that exploration of unknown edges takes place only at the for-loop of procedure special-explore, and in procedure explore-only-child. An edge is explored in a 1-phase (probe) only if it is the last one unknown (therefore a green edge). If this is the case then, since all other edges have been explored two by two, it means that the total number of edges in the tree is odd.

Let $e_g = (u_g, v_g)$ be a green edge of the tree (v_g is a leaf). Once edge e_g is explored, no agent will traverse it again. So in every case the number of traversals of a green edge is 2.

Algorithm Tree.
explore(s)

Procedure explore(v).
for every pair of unknown edges $(v,x),(v,y)$ incident to v **do**
split(x,y);
end for
if there is only one remaining unknown edge (v,z) incident to v **then**
probe(z);
end if
if every edge is explored **then**
repeat walk(s) **until** all remaining agents are at s
else
next := relocate(v);
explore(*next*)
end if

Let $e_r = (u_r, v_r)$ be a red edge of the tree (u_r, v_r are internal nodes and each of them has at least 2 children). Edge e_r is explored always during a 2-phase (split). The number of traversals of the edge e_r during its exploration is 2. Further traversals of e_r are done as follows.

Both agents traverse e_r from u_r to v_r in exactly one of the following ways:

- during a relocation (case 1.1),
- during a split (case 2.1),
- during a split (case 2.2.1).

Both agents traverse e_r from v_r to u_r in exactly one of the following ways:

- during a relocation (case 1.2),
- during a split (case 2.2.1),
- during the return to s.

In all three latter cases, every edge in subtree $T(v_r)$ is explored. Therefore none of the agents will again traverse edge e_r. Thus, the total number of traversals of a red edge is 6.

Let ρ denote the number of red edges and γ the number of green edges. If the total number of edges is even, every phase is a split. In that case the time of our scheme is $3\rho + \gamma$ which is optimal (see Lemmas 5.2 and 5.3). If the total number of edges is an odd number, every phase except one is a split. In that case the time of our scheme is $3\rho + \gamma + 1$. The optimal time has to be also at least $3\rho + \gamma + 1$, since in any BHS-scheme in that case at least one edge has to be explored during a 1-phase. □

5.2. An approximation algorithm for trees

In this section we give an approximation algorithm with ratio $\frac{5}{3}$ for the black hole search problem, working for arbitrary trees (*i.e.*, an algorithm which produces a BHS-scheme whose time is at most 5/3 of the shortest-possible time, for every input).

The high-level description of Algorithm *Tree* is as follows. Let v be the meeting point of the two agents after a phase (initially $v = s$); the edges with upper node v are explored by calling procedure split until either all such edges are explored or there is at most one remaining unknown edge incident to v, which is explored by calling procedure probe; this is repeated for any child of v. Aside from procedures split and probe, it uses function relocate, defined in the previous section.

The time complexity of Algorithm Tree is clearly linear.

Lemma 5.6. *Let u be a node which is neither a leaf nor a middle of a branch of blue edges. Let d be the down degree of u. Let β be the number of branches of blue edges with upper node u, ρ the number of red edges with upper node u and γ the number of green edges with upper node u. Algorithm Tree spends at most $d + 4\beta + 2\rho$ time units if d is even, and $d + 1 + 4\beta + 2\rho$ time units if d is odd for the traversals of all the above edges.*

Proof. The exploration of all edges $e_j = (u, v_j)$, for $1 \leqslant j \leqslant d$, takes time d if d is even, and time $d + 1$ if d is odd. Moreover Algorithm Tree also spends 4 time units for the additional traversals of each branch of blue edges and 2 time units for the additional traversals of each red edge. Therefore the total time spent is $d + 4\beta + 2\rho$ for even d and $d + 1 + 4\beta + 2\rho$ for odd d. □

Theorem 5.7. *Algorithm Tree achieves a $\frac{5}{3}$ approximation ratio.*

Proof. If the tree consists of a single edge, then the ratio is one. Otherwise, suppose that the tree has k nodes $u_1, u_2, \ldots, u_k$ such that $\forall u_i \exists v_j$ $(e_{ij} = (u_i, v_j))$ is a red edge, a green edge or an upper blue edge in a branch of blue edges. In any case, $\forall u_i \neq s$ u_i has at least two descendants, hence (u_i', u_i), where u_i' is the parent of u_i, is a red edge. Thus there are at least $k - 1$ red edges in the tree. Let $d_i : i = 1, \ldots, k$ be the down degree of u_i. Suppose that $d_i : i = 1, \ldots, l$ is odd and $d_i : i = l + 1, \ldots, k$ is even. Let β_i be the number of branches of blue edges with upper node u_i, ρ_i the number of red edges with upper node u_i and γ_i the number of green edges with upper node u_i. We have $d_i = \beta_i + \rho_i + \gamma_i$.

According to Lemma 5.3, any BHS-scheme must spend at least $3\beta_i + 3\rho_i + \gamma_i$ time units on the traversals of all red edges, green edges and branches of blue edges with upper node u_i. Hence, in view of Lemma 5.6 the ratio between the time of our scheme and the fastest-possible scheme is at most:

$$\frac{\sum_{i=1}^{l}(d_i + 1 + 4\beta_i + 2\rho_i) + \sum_{i=l+1}^{k}(d_i + 4\beta_i + 2\rho_i)}{\sum_{i=1}^{k}(3\beta_i + 3\rho_i + \gamma_i)} = \frac{\sum_{i=1}^{k}(5\beta_i + 3\rho_i + \gamma_i) + l}{\sum_{i=1}^{k}(3\beta_i + 3\rho_i + \gamma_i)}.$$

The above ratio is $\leqslant \frac{5}{3}$ when $3l \leqslant 6\sum_{i=1}^{k}\rho_i + 2\sum_{i=1}^{k}\gamma_i$. Since $\sum_{i=1}^{k}\rho_i \geqslant k - 1$, this ratio is lower than or equal to $\frac{5}{3}$ when

$$6(k-1) + 2\sum_{i=1}^{k}\gamma_i \geqslant 3l. \tag{5.1}$$

If $k-1 \geqslant l$ (*i.e.*, there is at least one node of even down degree) then inequality (5.1) is true.

If $k-1 < l$ it means that $l = k$. This is the situation when every vertex u_i has an odd down degree. If $k \geqslant 2$, inequality (5.1) still holds. If $k = 1$ then there is no red edge ($u_1 = s$). As long as there are at least two green edges, inequality (5.1) is true. Otherwise one of the following holds.

- The tree consists of a block of β_1 branches of blue edges where β_1 is even, and one green edge. In this case the total number of edges in the tree is odd. Hence, in any BHS-scheme at least one edge must be explored in a 1-phase. We prove that any BHS-scheme has to spend at least $3\beta_1 + 2$ time units for all the traversals. According to Lemma 5.1 the total number of traversals needed is at least $6\beta_1 + 2$. At least 2 of the traversals are done during a 1-phase and require at least 2 time units. Therefore, in view of Lemma 5.2, the time needed in this case is at least $\frac{6\beta_1}{2} + 2 = 3\beta_1 + 2$.
  According to Lemma 5.6, the scheme produced by Algorithm Tree uses $d_1 + 1 + 4\beta_1 = 5\beta_1 + 2$ time units. Thus the ratio is at most $\frac{5\beta_1+2}{3\beta_1+2} \leqslant \frac{5}{3}$.
- The tree consists of a block of β_1 branches of blue edges where β_1 is odd. If $\beta_1 = 1$ then the ratio is one. Otherwise we prove that any BHS-scheme has to spend in this case at least $3\beta_1 + 1$ time units for all traversals.
  - If there is an edge in a branch which has been traversed by both agents during a phase then the total number of edge traversals in that branch is 8. Therefore, in view of Lemmas 5.1 and 5.2, the total number of traversals is at least $6(\beta_1 - 1) + 8$ and the time needed is at least $\frac{6\beta_1+2}{2} = 3\beta_1 + 1$.
  - Otherwise, if there is at least one edge that has been explored during a 1-phase then the total number of traversals done during *2-phases* is at most $6\beta_1 - 2$ by Lemma 5.1, while there are 2 traversals done in a 1-phase which requires 2 time units. Therefore, by Lemma 5.2, the time needed is at least $\frac{6\beta_1-2}{2} + 2 = 3\beta_1 + 1$.
  - The remaining case is that every edge is explored during a 2-phase and there is no edge which has been traversed by both agents during a phase. Since the number of upper edges in branches is odd, there must be a 2-phase ϕ during which an upper edge of a branch is explored together with a lower edge of another branch. The time needed for this phase is at least 4 time units since both agents cannot traverse the same edge. In view of Lemma 5.1 the total number of traversals in every phase except ϕ is at least $6(\beta_1 - 2) + 2 + 4$ (there is a branch on which only 2 traversals are done and a branch on which only 4 traversals are done). Hence, by Lemma 5.2 the time needed in this case is at least $\frac{6\beta_1-6}{2} + 4 = 3\beta_1 + 1$.

  According to Lemma 5.6, the time of the scheme produced by Algorithm Tree is $d_1 + 1 + 4\beta_1 = 5\beta_1 + 1$ time units. Thus, in all three cases the ratio is at most $\frac{5\beta_1+1}{3\beta_1+1} \leqslant \frac{5}{3}$.
  □

Notice that there exists a family of trees in which the approximation ratio achieved by Algorithm Tree is exactly 5/3. This family includes all trees which consist of an even number β of branches of blue edges (see Figure 4). According to Lemma 5.6, the time of the scheme produced by Algorithm Tree is $\beta + 4\beta = 5\beta$ for such a tree, while the fastest

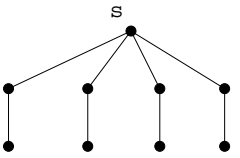

Figure 4. A tree in which the approximation ratio achieved by Algorithm Tree is exactly 5/3.

BHS-scheme for this tree requires exactly 3β time units (for example, all upper edges are explored two by two by calling procedure split, and then all lower edges are explored in the same way).

6. Conclusion

We have presented algorithms for the black hole search problem on trees. For arbitrary trees we gave a 5/3-approximation algorithm, and for two classes of trees (lines and trees all of whose internal nodes have at least 2 children) we gave optimal algorithms, *i.e.*, methods for constructing a shortest-possible black hole search scheme for any input in the class. The time complexity of all our algorithms is linear in the size of the input.

It remains open whether there exists a polynomial time algorithm to construct a fastest black hole search scheme for an arbitrary tree. After the publication of the conference version of this paper, it was proved in [7] that the problem of constructing a fastest black hole search scheme for an arbitrary planar graph is NP-hard. The authors also gave a $3\frac{3}{8}$ approximation algorithm for arbitrary graphs.

References

[1] Dobrev, S., Flocchini, P., Kralovic, R., Prencipe, G. Ruzicka, P. and Santoro, N. (2002) Black hole search by mobile agents in hypercubes and related networks. In *Proc. 6th Int. Conference on Principles of Distributed Systems* (OPODIS 2002), pp. 171–182.

[2] Dobrev, S., Flocchini, P., Prencipe, G. and Santoro, N. (2001) Mobile agents searching for a black hole in an anonymous ring. In *Proc. 15th International Symposium on Distributed Computing* (DISC 2001), Vol. 2180 of *Lecture Notes in Computer Science*, Springer, pp. 166–179.

[3] Dobrev, S., Flocchini, P., Prencipe, G. and Santoro, N. (2002) Searching for a black hole in arbitrary networks: Optimal Mobile Agents Protocols. In *Proc. 21st ACM Symposium on Principles of Distributed Computing* (PODC 2002), pp. 153–161.

[4] Dobrev, S., Flocchini, P., Prencipe, G. and Santoro, N. (2003) Multiple agents rendezvous on a ring in spite of a black hole. In *Proc. 7th Int. Conference on Principles of Distributed Systems* (OPODIS 2003), Vol. 3144 of *Lecture Notes in Computer Science*, Springer, pp. 34–46.

[5] Hohl, F. (1998) Time limited black box security: Protecting mobile agents from malicious hosts. In *Proc. Conf. on Mobile Agent Security* (1998), Vol. 1419 of *Lecture Notes in Computer Science*, Springer, pp. 92–113.

[6] Hohl, F. (2000) A framework to protect mobile agents by using reference states. In *Proc. 20th Int. Conf. on Distributed Computing Systems* (ICDCS 2000), pp. 410–417.

[7] Klasing, R., Markou, E., Radzik, T. and Sarracco, F. Hardness and approximation results for black hole search in arbitrary networks. *Theoret. Comput. Sci.*, to appear.

[8] Ng, S. and Cheung, K. (1999) Protecting mobile agents against malicious hosts by intention of spreading. In *Proc. Int. Conf. on Parallel and Distributed Processing and Applications* (PDPTA'99), Vol. II (H. Arabnia, ed.), pp. 725–729.
[9] Sander, T. and Tschudin, C. F. (1998) Protecting mobile agents against malicious hosts. In *Proc. Conf. on Mobile Agent Security* (G. Vigna, ed.), Vol. 1419 of *Lecture Notes in Computer Science*, Springer, pp. 44–60.
[10] Schelderup, K. and Ines, J. (1999) Mobile agent security: Issues and directions. In *Proc. 6th Int. Conf. on Intelligence and Services in Networks*, Vol. 1597 of *Lecture Notes in Computer Science*, Springer, pp. 155–167.
[11] Vitek, J. and Castagna, G. (1999) Mobile computations and hostile hosts. In *Proc. Journées Francophones des Langages Applicatifs* (JFLA 1999).